Constraint-based reachability

Arnaud Gotlieb, Tristan Denmat, Nadjib Lazaar

Constraint-based reachability

Arnaud Gotlieb
Certus Software V&V Center
SIMULA Research Laboratory
Lysaker, Norway
arnaud@simula.no

Tristan Denmat
INRIA Rennes Bretagne-Atlantique
Rennes, France
Tristan.Denmat@inria.fr

Nadjib Lazaar
LIRMM
Montpellier, France
lazaar@lirmm.fr

Iterative imperative programs can be considered as infinite-state systems computing over possibly unbounded domains. Studying reachability in these systems is challenging as it requires to deal with an infinite number of states with standard backward or forward exploration strategies. An approach that we call *Constraint-based reachability*, is proposed to address reachability problems by exploring program states using a constraint model of the whole program. The keypoint of the approach is to interpret imperative constructions such as conditionals, loops, array and memory manipulations with the fundamental notion of *constraint* over a computational domain. By combining constraint filtering and abstraction techniques, *Constraint-based reachability* is able to solve reachability problems which are usually outside the scope of backward or forward exploration strategies. This paper proposes an interpretation of classical filtering consistencies used in Constraint Programming as abstract domain computations, and shows how this approach can be used to produce a constraint solver that efficiently generates solutions for reachability problems that are unsolvable by other approaches.

1 Introduction

Modern automated program verification can be seen as the convergence of three distinct approaches, namely Software Testing, Model-Checking and Program Proving. Even if the general verification problems are often undecidable, investigations on these approaches have delivered the most efficient automated techniques to show that a given property is satisifed or not by all the reachable states of an infinite-state system.

Several authors have advocated the usage of *constraints* to represent an infinite set of states and the usage of constraint solvers to efficiently address reachability problems [6, 13, 16, 4]. In automated program verification problems, the goal is to find a state of the program which violates a given safety property, i.e., an *unsafe state*. Two distinct strategies have been investigated to explore programs with constraints, namely the forward analysis and the backward analysis strategies. In forward analysis, a set of reachable states is explored by computing the transition from the initial states of a program to the next states in forward way. If an unsafe state is detected to belong to the set of reachable states during this exploration then a property violation is reported. In backward analysis, states are computed from an hypotetical unsafe state in a backward way with the hope to discover that one of those is actually an initial state. One advantage of backward analysis over forward analysis is its usage of the targeted unsafe state to refine the state search space. However, both strategies are quite powerful and have been implemented into several software model checkers based on constraint solving [25, 16] and automated test case generators [29, 18, 17, 8, 3].

In this paper, we present an integrated constraint-based strategy that can benefit from the strengths of both forward and backward analysis. The keypoint of the approach, that we have called *Constraint-Based Reachability (CBR)*, is to interpret imperative constructions such as conditionals, loops, array and

M.F. Atig, A. Rezine (Eds.): Infinity'12
EPTCS ??, 2013, pp. 25–43, doi:10.4204/EPTCS.??.4

memory manipulations with the fundamental notion of *constraint* over a computational domain. By combining constraint filtering and abstraction techniques, CBR is able to solve reachability problems which are usually outside the scope of backward or forward exploration strategies. A main difference is that CBR does not sequentially explore the execution paths of the program ; the exploration is driven by the constraint solver which picks-up the constraint to explore depending on the priorities that are attached to them. It is worth noticing that applying CBR to program exploration results in a semi-correct procedure only, meaning that there is no termination guarantee. CBR has been mainly applied in automatic test data generation for iterative programs [21, 22], programs that manipulate pointers towards named locations of the memory [23, 24], programs on dynamic data structures and anonymous locations [7], programs containing floating-point computations [5]. A major improvement of the approach was brought by the usage of Abstract Interpretation techniques to enrich the filtering capabilities of the constraints used to represent conditionals and loops [14, 15]. This approach permitted us to build efficient test data generator tools for a subset of C [19] and Java Bytecode [8].

The first contribution of this paper is the interpretation of classical filtering consistencies notions in terms of abstract domain computations. Constraint filtering is the main approach behind the processing of constraints in a finite domains constraint solver. We show in general the existence of tight links between classical filtering techniques and abstract domain computations that were not pointed out elsewhere. We also give the definition of a new consistency filtering inspired from the Polyhedral abstract domain, as consequence of these links.

The second contribution is the description of a special constraint handling any iterative construction. The constraint *w* captures iterative reasoning in a constraint solver and as such, is able to deduce information which is outside the scope of any pure forward or backward abstract analyzer. Its filtering capabilities combines both constraint reasoning and abstract domain computations in order to propagate informations to the rest of the constraint system. In this paper, we focus on the theoretical foundations of the constraints, while giving examples of its usage for test case generation over iterative programs.

Outline of the paper. The rest of the paper is organized as follows. Sec.2 introduces the necessary background in Abstract Interpretation to understand the contributions of the paper. Sec.3 establishes the link between classical constraint filtering and abstract domain computations. Sec.4 describes the theoretical foundation of the *w* constraint for handling iterative constructions while Sec.5 concludes the paper.

2 Background

Abstract Interpretation (AI) is a theoretical framework introduced by Cousot and Cousot in [10] to manipulate abstractions of program states. An abstraction can be used to simplify program analysis problems otherwise not computable in realistic time, to manageable problems more easily solvable. Instead of working on the concrete semantics of a program[1], AI computes results over an abstract semantics allowing so to produce over-approximating properties of the concrete semantics. In the following we introduce the basic notions required to understand AI.

Definition 1 (Partially ordered set (poset)) *Let $\sqsubseteq$ be a partial order law, then the pair $(\mathscr{D},\sqsubseteq)$ is called a poset iff*

$$\begin{array}{ll} \forall x \in \mathscr{D}, x \sqsubseteq x & \textit{(reflexive)} \\ \forall x,y \in \mathscr{D}, x \sqsubseteq y \wedge y \sqsubseteq x \implies x = y & \textit{(anti-symmetry)} \\ \forall x,y,z \in \mathscr{D}, x \sqsubseteq y \wedge y \sqsubseteq z \implies x \sqsubseteq z & \textit{(transitive)} \end{array}$$

[1] Program semantics captures formally all the possible behaviours of a program.

Definition 2 (Complete lattice) *A complete lattice is a 4-tuple* $(\mathscr{D}, \sqsubseteq, \bigsqcup, \bigsqcap)$ *such that*

- $(\mathscr{D}, \sqsubseteq)$ *is a poset*
- $\bigsqcup$ *is a upper bound:* $\forall \mathscr{S} \subseteq \mathscr{D}$*, we have*

$$\forall x \in \mathscr{S}, x \sqsubseteq \bigsqcup \mathscr{S}$$
$$\forall y \in \mathscr{D}, (\forall x \in \mathscr{S}, x \sqsubseteq y) \implies \bigsqcup \mathscr{S} \sqsubseteq y$$

- $\bigsqcap$ *is a lower bound:* $\forall \mathscr{S} \subseteq \mathscr{D}$*, we have*

$$\forall x \in \mathscr{S}, \bigsqcap \mathscr{S} \sqsubseteq x$$
$$\forall y \in \mathscr{D}, (\forall x \in \mathscr{S}, y \sqsubseteq x) \implies y \sqsubseteq \bigsqcap \mathscr{S}$$

Complete lattices have a single smallest element $\bot = \bigsqcap \mathscr{D}$ and a single greatest element $\top = \bigsqcup \mathscr{D}$. Program semantics can usually be expressed as the least fix point of a monotonic and continuous function. A function f from a complete lattice $(\mathscr{D}, \sqsubseteq, \bigsqcup, \bigsqcap)$ to itself is monotonic iff $\forall l_1, l_2 \in \mathscr{D}, l_1 \sqsubseteq l_2 \implies f(l_1) \sqsubseteq f(l_2)$. It is continuous iff $\forall \mathscr{S} \subseteq \mathscr{D}, f(\bigsqcup \mathscr{S}) = \bigsqcup_{s \in \mathscr{S}}(f(s))$ and $f(\bigsqcap \mathscr{S}) = \bigsqcap_{s \in S}(f(s))$.
The following Theorem guarantees the existence of the fix points of a monotonic function.

Theorem 1 (Knaster-Tarski) *In a complete lattice* $(\mathscr{D}, \sqsubseteq, \bigsqcup, \bigsqcap)$*, for all monotonic functions* $f : \mathscr{D} \to \mathscr{D}$*,*

- *the least fix point of* f *(i.e.,* $lfp(f)$*) exists and* $lfp(f) = \bigsqcap \{x \mid f(x) \sqsubseteq x\}$
- *the greatest fix point of* f *(i.e.,* $gfp(f)$*) exists and* $gfp(f) = \bigsqcup \{x \mid f(x) \sqsubseteq x\}$

In addition, when the functions are continuous, these fix points can be computed using an algorithm derived from the following theorem:

Theorem 2 (Kleene) *In a complete lattice* $(\mathscr{D}, \sqsubseteq, \bigsqcup, \bigsqcap)$*, for all monotonic and continuous functions* $f : \mathscr{D} \to \mathscr{D}$*, the least fix point of* f *is equal to* $\bigsqcup \{f^n(\bot) \mid n \in \mathbb{N}\}$ *and the greatest fix point of* f *is equal to* $\bigsqcap \{f^n(\top) \mid n \in \mathbb{N}\}$

As $\bot, f(\bot), \ldots, f^n(\bot), \ldots$ is an increasing suite, we get $\bigsqcup \{f^n(\bot) \mid n \leq k\} = f^k(\bot)$. Hence, $\text{lfp}(f) = \lim_{k \to +\infty} f^k(\bot)$ and $\text{gfp}(f) = \lim_{k \to +\infty} f^k(\top)$.
For reaching the least fix point of a monotonic and continuous function in a complete lattice, it suffices to iterate f from $\bot$ until a fix point is reached.
Let $(\mathscr{D}, \sqsubseteq, \bigsqcup, \bigsqcap)$ be a complete lattice called the *concrete lattice* and f a function that defines some concrete semantics over this lattice, let $(\mathscr{D}^\sharp, \sqsubseteq^\sharp)$ be a poset called the *abstract poset*, and $f^\sharp : \mathscr{D}^\sharp \to \mathscr{D}^\sharp$ be a continuous function, then *Abstract Interpretation* aims at computing a fix point of $f^\sharp$ in order to over-approximate the computation performed by f.
Depending on whether the abstract poset is a complete lattice or not, we have distinct theoretical results regarding the abstraction. Proofs of the following theorems can be found in [11].

Galois connection When the abstract poset is a complete lattice, the notion of *Galois connection* is available to link the abstract computations with the concrete lattice.

Definition 3 (Galois connection) *Let* $(\mathscr{D}, \sqsubseteq, \bigsqcup, \bigsqcap)$ *and* $(\mathscr{D}^\sharp, \sqsubseteq^\sharp, \bigsqcup^\sharp, \bigsqcap^\sharp)$ *be two complete lattices, then a pair of functions* $\alpha : \mathscr{D} \to \mathscr{D}^\sharp$ *and* $\gamma : \mathscr{D}^\sharp \to \mathscr{D}$ *is a Galois connection iff* $\forall x \in \mathscr{D}, \forall y \in \mathscr{D}^\sharp, \alpha(x) \sqsubseteq^\sharp y \iff x \sqsubseteq \gamma(y)$ *noted:*

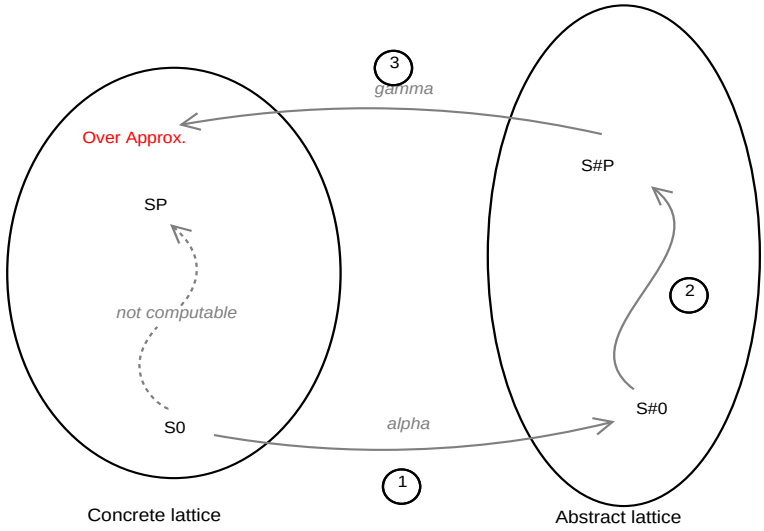

Figure 1: Static approximations of fixed point computations in complete lattices

$$(\mathscr{D},\sqsubseteq,\bigsqcup,\bigsqcap) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (\mathscr{D}^\sharp,\sqsubseteq^\sharp,\bigsqcup^\sharp,\bigsqcap^\sharp)$$

Next definition establishes the correction property of an analysis.

Definition 4 (Sound approximation) *Let* $(\mathscr{D},\sqsubseteq,\bigsqcup,\bigsqcap) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (\mathscr{D}^\sharp,\sqsubseteq^\sharp,\bigsqcup^\sharp,\bigsqcap^\sharp)$ *be a Galois connection, then a function* $f^\sharp : \mathscr{D}^\sharp \to \mathscr{D}^\sharp$ *is a sound approximation of* $f : \mathscr{D} \to \mathscr{D}$ *iff*

$$\forall y \in \mathscr{D}^\sharp, f \circ \gamma(y) \sqsubseteq \gamma \circ f^\sharp(y)$$

Consquently, we have the following notion:

Theorem 3 (Smallest sound approximation) *Let* $(\mathscr{D},\sqsubseteq,\bigsqcup,\bigsqcap) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (\mathscr{D}^\sharp,\sqsubseteq^\sharp,\bigsqcup^\sharp,\bigsqcap^\sharp)$ *be a Galois connection, and a function* $f : \mathscr{D} \to \mathscr{D}$*, then the smallest sound approximation of* f *is* $\alpha \circ f \circ \gamma$

This theorem implies that any function greater than $\alpha \circ f \circ \gamma$ is a sound approximation of f and the following theorem characterizes the results of fixpoint computations:

Theorem 4 (Fixpoint computations with sound approximation) *Let* $(\mathscr{D},\sqsubseteq,\bigsqcup,\bigsqcap) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (\mathscr{D}^\sharp,\sqsubseteq^\sharp,\bigsqcup^\sharp,\bigsqcap^\sharp)$ *be a Galois connection, let* $f^\sharp : \mathscr{D}^\sharp \to \mathscr{D}^\sharp$ *and* $f : \mathscr{D} \to \mathscr{D}$ *be two monotonic functions such that* $f^\sharp$ *is a sound approximation of* f*, then, we have:*

$$lfp(f) \sqsubseteq \gamma(lfp(f^\sharp)) \text{ and}$$
$$gfp(f) \sqsubseteq \gamma(gfp(f^\sharp))$$

Intuitively, this theorem gives a process to compute an over-approximation by Abstract Interpretation, as shown in Fig 1. The left part shows the concrete lattice where the concrete computation of f is performed starting from initial state S_0. The right part shows the abstract lattice that is used to over-approximate the computation. This computation is undertaken in three steps:

- initial state abstraction;
- fixpoint computation in the abstract lattice;
- result concretization.

Without Galois connection When the abstract lattice is not complete, there does not exist necessarily a best abstraction for all elements of the concrete lattice. The notion of Galois connection is no more available and the abstract lattice is just linked with the concrete lattice through a monotonic function $\gamma : \mathscr{D}^\sharp \to \mathscr{D}$. The definition of sound approximation needs to be adapted:

Definition 5 (Sound approximation without a Galois connection) *Let $(\mathscr{D}, \sqsubseteq)$ and $(\mathscr{D}^\sharp, \sqsubseteq^\sharp)$ be two posets, let $\gamma : \mathscr{D}^\sharp \to \mathscr{D}$ be a monotonic function and $f : \mathscr{D} \to \mathscr{D}$ a function, then the function $f^\sharp : \mathscr{D}^\sharp \to \mathscr{D}^\sharp$ is a sound approximation of f iff*

$$\forall x \in \mathscr{D}^\sharp, f \circ \gamma(x) \sqsubseteq \gamma \circ f^\sharp(x)$$

In such an (not complete) abstract lattice, nothing guarantees the existence of the least fix point: $\mathrm{lfp}(f)$ is not necessarily approximated by $\mathrm{lfp}(f^\sharp)$. However, any fix point of $f^\sharp$ can be used:

Theorem 5 *Let $(\mathscr{D}, \sqsubseteq, \bigsqcup, \bigsqcap)$ be a complete lattice, and $(\mathscr{D}^\sharp, \sqsubseteq^\sharp)$ be a poset, let $\gamma : \mathscr{D}^\sharp \to \mathscr{D}$, $f : \mathscr{D} \to \mathscr{D}$ and $f^\sharp : \mathscr{D}^\sharp \to \mathscr{D}^\sharp$ be three monotonic functions then if $f^\sharp$ is a sound approximation of f, then we have:*

$$\forall x \in \mathscr{D}^\sharp, f^\sharp(x) = x \implies lfp(f) \sqsubseteq \gamma(x)$$

Next theorem is useful to compute an over-approximation of $\mathrm{gfp}(f)$ when the lattice is not complete:

Theorem 6 *Let $(\mathscr{D}, \sqsubseteq, \bigsqcup, \bigsqcap)$ be a complete lattice, let $(\mathscr{D}^\sharp, \sqsubseteq^\sharp)$ be a poset with a greatest element $\top$ and let $\gamma : \mathscr{D}^\sharp \to \mathscr{D}$, $f : \mathscr{D} \to \mathscr{D}$ and $f^\sharp : \mathscr{D}^\sharp \to \mathscr{D}^\sharp$ be three monotonic functions, then*
if $f^\sharp$ is a sound approximation of f and a is an element of $\mathscr{D}^\sharp$ such that there exists k such as $a = f^{\sharp k}(\top)$, then

$$gfp(f) \sqsubseteq \gamma(a)$$

Consequently, when the abstract lattice is not complete, instead of abstracting the initial state, one selects an element of the abstract lattice that over-approximates the initial state. And, a fix point is computed in the abstract lattice from this element. The fix point is still an over-approximation of the concrete semantics.

2.1 Examples of abstract domains

In this section, we briefly describe two abstract domains: the Interval [12] and the Polyhedral [11] domains.

2.1.1 The Interval abstract domain

Interval analysis aims at approximating a set of values by an interval of possible values. If $\mathscr{I}_\mathbb{N} = \{[a,b] \mid a,b \in \mathbb{N} \cup \{-\infty, +\infty\}\}$, then the Interval abstract domain is the Cartesian product $\mathscr{I}_\mathbb{N} \times ... \times \mathscr{I}_\mathbb{N}$ equipped with inclusion, union and intersection over intervals. This abstract domain is a complete lattice. State abstraction is performed by computing an interval that over-approximates the set of possible values for each variable. If the concrete state is an unbounded set of tuples $\{(x_{11}, \ldots, x_{n1}), (x_{12}, \ldots, x_{n2}), \ldots\}$

then:

$$\alpha(\{(x_{11},\ldots,x_{n1}),(x_{12},\ldots,x_{n2}),\ldots\}) = ([m_1,M_1],\ldots,[m_n,M_n])$$
$$\text{with } m_i = \begin{cases} min_j(x_{ij}) \text{ if it exists} \\ -\infty \text{ else} \end{cases} \quad \text{and } M_i = \begin{cases} max_j(x_{ij}) \text{ if it exists} \\ +\infty \text{ else} \end{cases}$$

The concretization of an abstract state is obtained by computing the Cartesian product of the intervals. These functions define a Galois connection between the concrete domain and the abstract domain of intervals.
The approximation of transfert functions is realized by using their structure and classical results from Interval Analysis [27]. For example, functions $[x \leq y]$ and $[x = y+z]$ are abstracted by the following (sound) approximations: $[x \leq y]^\sharp : ([a,b],[c,d]) \rightarrow ([a,min(b,d)],[max(a,c),d]))$ and $[x = y+z]^\sharp : ([a,b],[c,d],[e,f]) \rightarrow ([c+e,d+f] \cap^\sharp [a,b],[a-f,e-b] \cap^\sharp [c,d],[a-d,c-b] \cap^\sharp [e,f])$.

2.1.2 The Polyhedral abstract domain

In Polyhedral analyses, each concrete state is abstracted by a conjunction of linear constraints that defines a convex polyhedron. Indeed, a *convex polyhedron* is a region of an n-dimensional space that is bounded by a finite set of hyperplanes $x \in \mathbb{R}^n | ax \geq c$ where $a \in \mathbb{R}^n$ and $c \in \mathbb{R}$. The abstract lattice equiped with inclusion, convex hull[2], and intersection of polyhedra is not a complete lattice as there is no upper bound to the convex union of all the convex polyhedra that can be written in a circle.
Abstract functions can be defined to deal with polyhedra. For example:

$$[x \geq y]^\sharp(\{z \leq x+y\}) = \{z \leq x+y, y \leq x\} \quad (1)$$
$$[x > y]^\sharp(\{x \leq y\}) = \{0 = 1\} \quad (2)$$
$$[x = y*z]^\sharp(\{1 \leq y \leq 10\}) = \{x \leq z, x \leq 10*z\} \quad (3)$$

If the expression is a linear condition, then it is just added to the polyhedron (case 1). If the expression is contradictory with the current polyhedron, then it is reduced to $1 = 0$ meaning that there is no abstract (and concrete) state in the approximation (case 2). If the expression is non-linear, then a linear approximation is derived when available and added to the polyhedron (case 3).

3 Filtering consistencies as abstract domain computations

As noticed by Apt [1], constraint propagation algorithms can be seen as instances of algorithms that deal with chaotic iteration. In this context, chaotic means fair application of propagators until saturation. In this section, we elaborate on a bridge between two unrelated notions: filtering consistencies and abstract domains. In particular, we show that arc– and bound– consistency are instances of chaotic iterations over two distinct abstract domains. Classical AI notions of sound approximation and abstract domain computations, not used in [1], allows to show that filtering consistencies compute sound over-approximations of the solutions set of a constraint system. Thanks to the bridge, we also propose new filtering consistency algorithms based on the polyhedral abstract domain.

[2] The union of two polyhedra is not a polyhedron, this is the reason why convex hull or any relaxation of it must be employed.

3.1 Notations

Let Z be the set of integers and $\mathscr{V}$ be a finite set of integer variables, where each variable x in $\mathscr{V}$ is associated with a finite domain $D(x)$. The *domain* $\mathscr{D}$ is the Cartesian product of each variable domain: $D(x_1) \times \ldots \times D(x_m)$ and $\mathscr{P}(\mathscr{D})$ denotes the powerset of $\mathscr{D}$. $inf_{\mathscr{D}}x$ and $sup_{\mathscr{D}}x$ denote respectivelly the inferior and the superior bounds of $D(x)$ in $\mathscr{D}$. A constraint c is a relation between variables of $\mathscr{V}$. The language of (elementary) constraints is built over arithmetical operators $\{+,-,*, ...\}$ and relational operators $\{<,\leq,>,\geq,=,\neq,...\}$ but any relation over a subset of $\mathscr{V}$ can be considered. Let $vars(c)$ be the function that returns the variables of $\mathscr{V}$ appearing in a constraint c. A *valuation* σ is a mapping of variables to values, noted $\{x_1 \mapsto d_1, ..., x_n \mapsto d_n\}$. *CS* denotes a constraint system *CS*, i.e., a finite set of constraints.

3.2 Exact filtering

Let $\{c_1, .., c_m\}$ be a *CS* over $\{x_1, .., x_n\}$ and let $\mathscr{D} = D(x_1) \times .. \times D(x_n)$, then the solution-set of *CS* is an element of $\mathscr{P}(\mathscr{D})$, noted $sol(CS)$.
The *exact filtering operator* of a constraint c_i is computed with the function $f_i : \mathscr{P}(\mathscr{D}) \to \mathscr{P}(\mathscr{D})$ which maps an element $S \in \mathscr{P}(\mathscr{D})$ to $f_i(S) = \{s \mid s \in S \wedge c_i(s)\}$. The exact filtering operator of c_i removes *all* the tuples of $\mathscr{D}$ that violate c_i. Hence, by using an iterating procedure, it permits to compute $sol(CS)$: if $f_C = f_1 \circ \ldots \circ f_m$ then $sol(C) = \text{gfp}(f_C)$. By noticing that f_C is continuous (as each f_i is continuous) and monotonic and thanks to Theorem 2 we get $sol(CS) = \lim_{k \to +\infty} {f_C}^k(\mathscr{D})$.

Example 1 *Consider* $CS = \{x \neq y, y \neq z, z \neq x\}$ *where* $x \in 1..2, y \in 1..2, z \in 1..2$. *The exact filtering operator associated with* $x \neq y$ *will remove the tuples* $(1,1,1),(1,1,2),(2,2,1),(2,2,2)$ *from* $\{1,2\} \times \{1,2\} \times \{1,2\}$. *Iterating over all the constraints of CS will eventually exhibit the inconsistency of this example.*

In fact, this shows that exact filtering of a CS over $D(x_1) \times .. \times D(x_n)$ can be reached if one computes over a complete lattice built over the set of possible valuations: $(\mathscr{P}(D(x_1) \times .. \times D(x_n)), \subseteq, \bigcup, \bigcap)$. This lattice will be called the *concrete lattice* in the rest of the paper. Of course, computing over the concrete lattice is usually unreasonable, as it requires to examine every tuple of the Cartesian product $D(x_1) \times .. \times D(x_n)$ w.r.t. consistency of each constraint.

3.3 Domain-consistency filtering

For binary constraint systems, the most successful local consistency filtering is arc-consistency, which ensures that every value in the domain of one variable has a support in the domain of the other variable. The standard extension of arc-consistency for constraints of more than two variables is domain-consistency (also called hyper-arc consistency [26]). Roughly speaking, the abstraction that underpins domain-consistency filtering aims at considering each variable domain separately, instead of considering the Cartesian product of each individual domain. More formally,

Definition 6 (Domain-consistency) *A domain* $\mathscr{D}$ *is domain-consistent for a constraint c where* $vars(c) = \{x_1, .., x_n\}$ *iff for each variable* x_i, $1 \leq i \leq n$ *and for each* $d_i \in D(x_i)$ *there exist integers* d_j *with* $d_j \in D(x_j)$, $1 \leq j \leq n, j \neq i$ *such that* $\sigma = \{x_1 \mapsto d_1, .., x_n \mapsto d_n\}$ *is an integer solution of c.*

Consider the domains $\mathscr{D} = D(x_1) \times .. \times D(x_n)$ and $\mathscr{D}^\sharp_{arc} = \mathscr{P}(D(x_1)) \times \ldots \times \mathscr{P}(D(x_n))$ and the abstraction function $\alpha_{arc} : \mathscr{P}(\mathscr{D}) \to \mathscr{D}^\sharp_{arc}$ which maps $S \in \mathscr{P}(\mathscr{D})$ to

$$\alpha_{arc}(S) = (\{x_1 \mid x \in S\}, \ldots, \{x_n \mid x \in S\})$$

The concretization function is a function $\gamma_{arc} : \mathscr{D}^{\sharp}_{arc} \rightarrow \mathscr{P}(\mathscr{D})$ such that

$$\gamma_{arc}((S_1,\ldots,S_n)) = S_1 \times \ldots \times S_n$$

If $\sqsubseteq^{\sharp}_{arc}$, $\bigsqcup^{\sharp}_{arc}$ and $\bigsqcap^{\sharp}_{arc}$ denote respectivelly the inclusion, union and intersection of two tuples of sets, then we got the following Galois connection:

$$(\mathscr{P}(\mathscr{D}),\subseteq,\bigcup,\bigcap) \underset{\alpha_{arc}}{\overset{\gamma_{arc}}{\leftrightarrows}} (\mathscr{D}^{\sharp}_{arc},\sqsubseteq^{\sharp}_{arc},\bigsqcup^{\sharp}_{arc},\bigsqcap^{\sharp}_{arc})$$

The proof follows comes the monotonicity of the projection and Cartesian product. From Theorem 3, we get:

Definition 7 *The best sound approximation of the exact filtering operator f_i is*

$$f^{\sharp}_{i\_arc} \overset{\text{def}}{=} \alpha_{arc} \circ f_i \circ \gamma_{arc}$$

Theorem 7 *Let p be a filtering operator associated with constraint c_i, then p computes domain-consistency iff $p = f^{\sharp}_{i\_arc}$.*

This theorem implies that domain-consistency is the strongest property that can be guaranteed by a filtering operator using the abstraction α_{arc}. A proof is given in the Appendix of the paper.
Let us consider now the function $f^{\sharp}_{arc}$ such that $f^{\sharp}_{arc} = f^{\sharp}_{1\_arc} \circ \ldots \circ f^{\sharp}_{n\_arc}$. As $f^{\sharp}_{arc}$ is a sound approximation of f_C then

$$sol(C) = \text{gfp}(f_C) \subseteq \gamma_{arc}(\text{gfp}(f^{\sharp}_{arc}))$$

This result shows if necessary that constraint propagation over domain-consistency filtering operators computes an over-approximation of the solution set of C.

3.4 Bound-consistency filtering

Following the same scheme, AI can be used to show the abstraction that underpins constraint propagation with bound-consistency filtering (also called interval-consistency). But, firstly, let us recall the definition of bound-consistency we consider in this paper, as several definitions exist in the literature [9] :

Definition 8 (Bound-consistency) *A domain $\mathscr{D}$ is bound-consistent for a constraint c where $vars(c) = \{x_1,..,x_n\}$ iff for each variable x_i, $1 \leq i \leq n$ and for each $d_i \in \{inf_{\mathscr{D}}x_i, sup_{\mathscr{D}}x_i\}$ there exist integers d_j with $inf_{\mathscr{D}}x_j \leq d_j \leq sup_{\mathscr{D}}x_j$, $1 \leq j \leq n, j \neq i$ such that $\sigma = \{x_1 \mapsto d_1,..,x_n \mapsto d_n\}$ is an integer solution of c.*

Roughly speaking, this approximation considers only the bounds of the domain of each variable and approximates each domain with an interval. Let $\mathscr{I}(S) = [min(S), max(S)]$ be the smallest interval that contains all the elements of a finite set of integers S. Similarly, $\mathscr{I}^{-1}(I)$ denotes the set of integers of an interval I : $\mathscr{I}^{-1}([a,b]) = \{x \in \mathbb{Z} \mid a \leq x \leq b\}$.
The abstract domain we consider for bound-consistency is $\mathscr{D}^{\sharp}_{bound} = \mathscr{I}(\mathscr{P}(D(x_1))) \times \ldots \times \mathscr{I}(\mathscr{P}(D(x_n)))$. Given a tuple of sets $(S_1,\ldots,S_n)$ and a tuple of intervals $(I_1,\ldots,I_n)$, we consider the functions α_{inter} and γ_{inter} such that:

$$\alpha_{inter}(S_1,\ldots,S_n) = (\mathscr{I}(S_1),\ldots,\mathscr{I}(S_n))$$
$$\gamma_{inter}(I_1,\ldots I_n) = (\mathscr{I}^{-1}(I_1),\ldots,\mathscr{I}^{-1}(I_n))$$

Let $\alpha_{bound} : \mathscr{P}(\mathscr{D}) \to \mathscr{D}^{\sharp}_{bound}$ be an abstraction function such that

$$\alpha_{bound} = \alpha_{inter} \circ \alpha_{arc}$$

and $\gamma_{bound} : \mathscr{D}^{\sharp}_{bound} \to \mathscr{P}(\mathscr{D})$ be a concretization function such that

$$\gamma_{bound} = \gamma_{arc} \circ \gamma_{inter}$$

If $\sqsubseteq^{\sharp}_{bound}$, $\bigsqcup^{\sharp}_{bound}$ and $\bigsqcap^{\sharp}_{bound}$ respectively denote inclusion, union and intersection of intervals (component by component) then we get the following Galois connection:

$$(\mathscr{P}(\mathscr{D}), \subseteq, \bigcup, \bigcap) \underset{\alpha_{bound}}{\overset{\gamma_{bound}}{\leftrightarrows}} (\mathscr{D}^{\sharp}_{bound}, \sqsubseteq^{\sharp}_{bound}, \bigsqcup^{\sharp}_{bound}, \bigsqcap^{\sharp}_{bound})$$

Let $f^{\sharp}_{i\_bound}$ be the most accurate sound approximation of f_i, then we get:

$$\begin{aligned} f^{\sharp}_{i\_bound} &= \alpha_{bound} \circ f_i \circ \gamma_{bound} \\ &= \alpha_{inter} \circ f^{\sharp}_{i\_arc} \circ \gamma_{inter} \end{aligned}$$

Theorem 8 *If p is a filtering operator associated to constraint c_i, then p computes bound-consistency iff $p = f^{\sharp}_{i\_bound}$.*

This theorem, proved in Appendix, implies that bound-consistency is the strongest property that can be reached with an operator based on the α_{bound} abstraction.
Consider now the function $f^{\sharp}_{bound}$ such that $f^{\sharp}_{bound} = f^{\sharp}_{1\_bound} \circ \ldots \circ f^{\sharp}_{n\_bound}$. As $f^{\sharp}_{bound}$ is a sound approximation of f_C, then

$$sol(C) = \text{gfp}(f_C) \subseteq \gamma_{bound}(\text{gfp}(f^{\sharp}_{bound}))$$

This result shows if necessary that constraint propagation based on bound-consistency computes a sound over-approximation of the solution set of C. In addition, as $f^{\sharp}_{bound}$ is also a sound over-approximation of $f^{\sharp}_{arc}$, then

$$\gamma_{arc}(\text{gfp}(f^{\sharp}_{arc})) \subseteq \gamma_{bound}(\text{gfp}(f^{\sharp}_{bound}))$$

meaning that filtering with bound-consistency provides an over-approximation of the results given by a filtering with domain-consistency.

3.5 New filtering consistencies based on abstract domains

In the previous section, classical filtering consistencies are interpreted in terms of abstract domain computations. In this section, we propose a new filtering consistency based on the Polyhedral abstract domain [11].

3.5.1 Linear relaxations

When non-linear constraints are involved in a constraint store, approximating them with linear constraints is natural in order to benefit from powerful Linear Programming techniques. These techniques can be used to check the satisfiability of the constraint store when the approximation is sound. If the approximate constraint system is unsatisfiable so is the non-linear constraint system. But, in the context

of optimization problems, the approximation can also be used to prune current bounds of the function to optimize.

Another form of approximation comes from the domain in which the computation occurs. A linear problem over integers can be relaxed in the domain of rationals or reals and solved within this domain. As the set of integers belongs to the rationals and reals, an integer solution of the relaxed problem is also a solution of the original integer problem, but the converse is false. In this paper, we will consider both kinds of approximations under the generic term of "linear relaxations".

Computing a linear relaxation of a constraint system *CS* aims at finding a set of linear constraints that characterizes an over-approximation of the solution set of *CS*. It is not unique but for trivial reasons, we are more interested in the tighter possible relaxations. The tightest linear relaxation is the convex hull of the solution set of *CS* but computing this relaxation is as hard as solving *CS*. For *CS* over finite domains, the problem is therefore NP_hard. Whenever a relaxation is computed by using the current bounds of variable domains, it is called *dynamic* and the consistencies presented in the rest of the section are compatible with dynamic linear relaxations.

3.5.2 Polyhedral-consistency filtering

Let *Poly* be the abstract domain of closed convex polyhedra with rational coefficients. As said previously, *Poly* is not a complete lattice, and then we cannot define a Galois connection between *Poly* and the lattice of the solutions. Nevertheless, the concretization function $\gamma_{poly} : Poly \rightarrow \mathscr{P}(\mathscr{D})$ can be defined as the function that returns the integer points of a given polyhedron:

$$\gamma_{poly}(S^\sharp) = \text{int\_sol}(S^\sharp)$$

Here, int_sol stands for the whole set of integer solutions of a set of linear constraints As $S^\sharp$ is bounded, $\gamma_{poly}(S^\sharp)$ is finite.

Without a Galois connection, we do not expect the polyhedral-consistency proposed in this section to be optimal w.r.t. the abstract domain. Hence, we only show that the filtering algorithm that computes this consistency is a sound approximation of the exact filtering operator.

Definition 9 *Let α_{box} be the following abstraction function*
$\alpha_{box} : \mathscr{D}^\sharp_{bound} \rightarrow Poly$ *such that*

$$\alpha_{box}(([a_1,b_1],\ldots,[a_m,b_m])) = \{a_1 \leq x_1 \leq b_1,\ldots,a_m \leq x_m \leq bm\}$$

and the concretization function $\gamma_{box} : Poly \rightarrow \mathscr{D}^\sharp_{bound}$:

$$\gamma_{box}(P) = \begin{cases} ([\lceil min(x_1,P)\rceil, \lfloor max(x_1,P)\rfloor],\ldots,[\lceil min(x_m,P)\rceil, \lfloor max(x_m,P)\rfloor]) \\ \quad if\ \forall i, \lceil min(x_i,P)\rceil \leq \lfloor max(x_i,P)\rfloor \\ \emptyset\ otherwise \end{cases}$$

where $\lfloor x \rfloor$ (resp. $\lceil x \rceil$) stands for the next smallest (resp. largest) integer of x, and $min(v,P)$ (resp. $max(v,P)$) computes the smallest (resp. largest) value of v corresponding to a point of P.
Both α_{box} and γ_{box} link the polyhedral abstract domain with the interval abstract domain. The abstraction function α_{box} maps a set of intervals into a polyhedron by adding two inequalities per variable, while the concretization function γ_{box} maps a polyhedron into a set of intervals by computing first the smallest hypercuboid containing the polyhedron and second the greatest hypercuboid with integer bounds. The behaviour of these two functions is illustrated in Fig. 2.

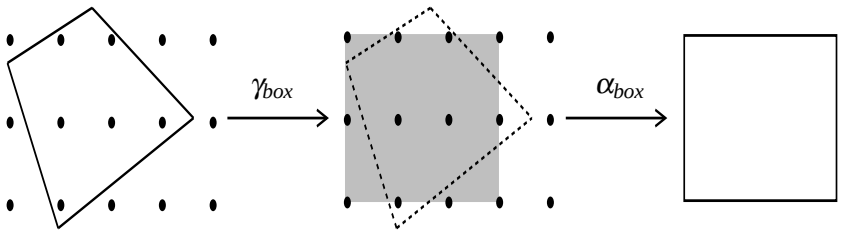

Figure 2: Connection between the Polyhedral and Interval abstract domains

Definition 10 (Polyhedral-consistency) *A domain $\mathscr{D}$ is polyhedral-consistent for a constraint c where $vars(c) = \{x_1,..,x_n\}$ iff for each variable x_i, $1 \leq i \leq n$ and for each $d_i \in \{inf_{\mathscr{D}}x_i, sup_{\mathscr{D}}x_i\}$ there exist rationals r_j with $inf_{\mathscr{D}}x_j \leq r_j \leq sup_{\mathscr{D}}x_j$, $1 \leq j \leq n, j \neq i$ such that $\sigma = \{x_1 \mapsto r_1,..,x_n \mapsto r_n\}$ is a (rational) solution of a linear relaxation of c.*

The rationale behind this definition is to benefit from efficient polyhedral techniques over the rationals to filter the variation domain of variables. Of course, interesting implementations of this filtering consistency should trade between efficiency and precision as integer linear constraint solving is costly (NP_hard problem) even for bounded domains. It is worth noticing that the definition depends on the quality of the underlying linear relaxation. On the one hand, a linear relaxation which over-approximate *c* by *True* (the whole search space) is useless while on the other hand a linear relaxation which exploits piecewise over-approximations of *c* is often too costly. We give examples of polyhedral-consistency filtering in function of various linear relaxations.

Example 2 *Consider the following CS: $z = x+y, z = x*y$, let c be the second constraint of CS: $c = (z = x*y)$ and let $\mathscr{D}$ be $x \in -7..10, y \in -7..10, z \in 3..10$.*
Note that $\mathscr{D}$ is bound-consistent for all the constraints of CS.
The simplest linear relaxation that can be considered is the one that ignores non-linear constraints. In this example, c is over-approximated by True and then $\mathscr{D}$ viewed as $x \geq -7, x \leq 10, y \geq -7, x \leq 10, z \geq 3, x \leq 10, z = x+y$ is then polyhedral-consistant w.r.t. this linear relaxation. Note that this approach can be generalized by associating a new fresh variable to the non-linear term $x*y$ with a domain computed using the bounds x and y. In this example, this does not help but it could help on other examples.
Another linear relaxation consists in building a polyhedron from the "bounds" of $x*y$ in $\mathscr{D} = x \in -7..10, y \in -7..10, z \in 3..10$. By considering the 2-dimensional polyhedron
$\{(1,10),(10,1),(-1,-7),(-7,-1)\}$ *we get that a linear relaxation of c in domain $\mathscr{D}$ is*
$11x - 8y + 69 \geq 0$
$-x - y + 11 \geq 0$
$-8x + 11y + 69 \geq 0$
$x + y + 8 \geq 0$
Filtering with the polyhedral-consistency, we get that $x \in -2..9, y \in -2..9, z \in 3..10$ where $D(x)$ and $D(y)$ have been pruned. These results can be easily computed using a Linear Programming tool and truncation operators. For example, using the clpq library of SICStus Prolog which implements a simplex over the rationals, the following request permits to compute the max bound of variable x:

```
{X >= -7, X =< 10, Y >= -7, Y =< 10, Z >= 3, Z =< 10, Z = X+Y,
11*X - 8*Y+ 69 >=0, -X - Y + 11 >= 0, -8*X + 11*Y +69 >= 0,
X + Y + 8 >=0}, sup(X, R).

R = 179/19   % then max bound of x is 9
```

Finally, we can automate the computation of linear relaxations of c by considering the following trivial constraints, which are always true for any x and y: $(x - inf_{\mathscr{D}}x)(y - inf_{\mathscr{D}}y) \geq 0$
$(x - sup_{\mathscr{D}}x)(y - inf_{\mathscr{D}}y) \leq 0$
$(x - inf_{\mathscr{D}}x)(y - sup_{\mathscr{D}}y) \leq 0$
$(x - sup_{\mathscr{D}}x)(y - sup_{\mathscr{D}}y) \geq 0$

By decomposing these constraints, using the original bounds of x,y,z *and replacing the quadratic term* $x*y$ *by z, we get:*
$7x + 7y + z + 49 \geq 0$
$10x - 7y - z + 70 \geq 0$
$-7x + 10y - z + 70 \geq 0$
$-10x - 10y + z + 100 \geq 0$
Filtering with the polyhedral-consistency, we get that $x \in -2..9, y \in -2..9, z \in 3..10$ *where* $D(x)$ *and* $D(y)$ *have been pruned. These domains are still bound-consistent but another tighter relaxation can be computed with these new bounds:*
$2X + 2Y + Z + 4 = 0$
$9X - 2Y - Z + 18 = 0$
$-2X + 9Y - Z + 18 = 0$
$-9X - 9Y + Z + 81 = 0$
and then filtering again permits to get that $x \in 0..8, y \in 0..8, z \in 3..10$*. Here, filtering by bound-consistency leads to prune the domains to:* $x \in 1..8, y \in 1..8, z \in 3..10$*. Then, by iterating these two process, we get the only solution to CS which is:* $x \in 2..2, y \in 2..2, z \in 4..4$*. This showed how dynamic linear relaxations can be used to solve a non-linear CS.*

4 The *w* constraint operator

In this section, we present the *w* constraint operator which captures iterative computations, and how it is processed by a constraint solver. The constraint operator has been introduced a long time ago in [21, 22] and was further refined using Abstract Interpretation (AI) techniques [14]. In the following, we recall its interface and semantics and show how fixed point computations can be used to filter inconsistant values of the underlying relation. We also explain how the Polyhedral abstract domain is used to approximate the fixed point computations.

4.1 *w* as a relation over memory states

The *w* operator captures a relation over three memory states that represent the state before, within and after the execution of an iterating statement. In this paper, we do not specify what a memory state is, or what the iterating statement is, as the approach is generic regarding the content of a memory state and the concrete syntax of the iterator. However, in order to ease the understanding, the reader can consider

a memory state to be a mapping between variables of the program to values. More complex examples of memory states in relation with w can be found in [7] and [8].
The relation w is expressed with the following syntax: $w(\mathscr{M}_1, \mathscr{M}_2, \mathscr{M}_3, Dec, Body)\}$ where $\mathscr{M}_1$ denotes the memory state before execution of the iteration, $\mathscr{M}_2$ denotes the memory state reached at the end of execution of the *Body*, while $\mathscr{M}_3$ denotes the state after execution, *Dec* is a boolean syntactical expression, and *Body* is a list of statements. This three-states consideration is inspired by the Static Single Assignment of a program [28]. If the state of $\mathscr{M}$ is irrelevant for a given computation, we simply write _. Note that *Body* may also contain other iterators, and thus w is meant to be a compositional operator. The semantics of w is the semantics of an iterating statement (i.e., repetitive application of *Body* over an input state, while *Dec* is true).
We note $w^n = \overbrace{w \circ w \ldots w}^{n}$ where $\circ$ is the application composition.

4.2 Background on w

As described in [22], the operational semantics of w within a constraint solver is expressed as a set of guarded-constraints: $\{(C_1 \longrightarrow C_2)_i\}_{1 \leq i \leq n}$. If C_1 is entailed by the constraint store then C_2 is added to it, and the relation w is solved. If C_1 is disentailed, then the guarded-constraint is discarded and no more considered in further analysis. Finally, if none of these (dis-) entailment deductions is possible, the guarded-constraint just suspends in the constraint store. The set of guarded-constraints is considered each time the constraint w awakes in the constraint store, so that it captures the essence of the iteration through rewriting in recursive calls. In addition, substitution of variables must be considered to faithfully represent the constraints in a w relation. $Dec_{\mathscr{M}_3 \leftarrow \mathscr{M}_1}$ simply denotes the constraint *Dec* where program variables from $\mathscr{M}_3$ have been substituted by the variables from $\mathscr{M}_1$. With these notations, the w relation is expressed as follows:
$w(Dec, \mathscr{M}_1, \mathscr{M}_2, \mathscr{M}_3, Body)$ iff

- $Dec_{\mathscr{M}_3 \leftarrow \mathscr{M}_1} \longrightarrow Body_{\mathscr{M}_3 \leftarrow \mathscr{M}_1} \wedge w(Dec, \mathscr{M}_2, \mathscr{M}_{new}, \mathscr{M}_3, Body_{\mathscr{M}_2 \leftarrow \mathscr{M}_{new}})$
- $\neg(Dec_{\mathscr{M}_3 \leftarrow \mathscr{M}_1}) \longrightarrow \mathscr{M}_3 = \mathscr{M}_1$
- $\neg(Dec_{\mathscr{M}_3 \leftarrow \mathscr{M}_1} \wedge Body_{\mathscr{M}_3 \leftarrow \mathscr{M}_1}) \longrightarrow \neg(Dec_{\mathscr{M}_3 \leftarrow \mathscr{M}_1}) \wedge \mathscr{M}_3 = \mathscr{M}_1$
- $\neg(\neg Dec_{\mathscr{M}_3 \leftarrow \mathscr{M}_1} \wedge \mathscr{M}_3 = \mathscr{M}_1) \longrightarrow Dec_{\mathscr{M}_3 \leftarrow \mathscr{M}_1} \wedge Body_{\mathscr{M}_3 \leftarrow \mathscr{M}_1} \wedge w(Dec, \mathscr{M}_2, \mathscr{M}_{new}, \mathscr{M}_3, Body_{\mathscr{M}_2 \leftarrow \mathscr{M}_{new}})$
- $join(Dec_{\mathscr{M}_3 \leftarrow \mathscr{M}_1} \wedge Body_{\mathscr{M}_3 \leftarrow \mathscr{M}_1} \wedge w(Dec, \mathscr{M}_2, \mathscr{M}_{new}, \mathscr{M}_3, Body_{\mathscr{M}_2 \leftarrow \mathscr{M}_{new}}), \neg(Dec_{\mathscr{M}_3 \leftarrow \mathscr{M}_1}) \wedge \mathscr{M}_3 = \mathscr{M}_1)$

The two former guarded-constraints implement forward analysis, by examining the entailment of *Dec*. Depending on the entailment of *Dec*, a recursive call to a new w is added to the constraint store. The two followings implement backward reasoning by examining the differences between the stores after and before execution of the iteration. Finally, the last operation, called *join*, is the most tricky one and implements union of stores in case of suspension of the operator. This *join* operation is realized iff none of the previous guarded-constraints has been solved. The rest of the Section is devoted to the presentation of this operator, which is implemented as an abstract operation over abstract domains.

4.3 Concrete fixed point computation

For a given w operator, let T be the following set:

$$T = \{(\mathscr{M}_i, \mathscr{M}_j) \mid \exists k \mid w^k(\mathscr{M}_i, \mathscr{M}_j, \_, Dec, Body)\}$$

T represents all pairs of memory states that are in relation through the w statement, but still, not all those

pairs can be considered as solutions of the relation, as some pairs can only be reached in temporary states of the execution. For this reason, we introduce the set Z_w:

$$Z_w = \{(\mathscr{M}_i, \mathscr{M}_j) \mid (\mathscr{M}_i, \mathscr{M}_j) \in T \wedge \mathscr{M}_j \in sol(\neg Dec)\}$$

where $sol(C)$ denotes the set of solutions of a constraint C.
T can be seen as the *least fixed point* of:

$$T^{i+1} = \{(\mathscr{M}_k, \mathscr{M}_j) \mid (T^i \wedge w(\mathscr{M}_k, \_, \mathscr{M}_j, Dec, Body))\} \cup T^i \quad (4)$$
$$T^0 = \{(\mathscr{M}_1, \mathscr{M}_1)\} \quad (5)$$

and Z_w can be computed by filtering the pairs of the fixed point.
For instance, considering $\mathscr{M}_1 = x \mapsto 0 \vee x \mapsto 1 \vee x \mapsto 2 \vee x \mapsto 3$ and $w(\mathscr{M}_1, \mathscr{M}_2, \mathscr{M}_3, x < 2, x = x + 1)$, and using the notation $(0,0)$ for denotating $(x \mapsto 0, x \mapsto 0)$, the fix point computation is as follows:

$$\begin{aligned}
T^0 &= \{(0,0),(1,1),(2,2),(3,3)\} \\
T^1 &= \{(0,1),(1,2)\} \cup T^0 = \{(0,0),(0,1),(1,1),(1,2),(2,2),(3,3)\} \\
T^2 &= \{(0,1),(0,2),(1,2)\} \cup T^1 = \{(0,0),(0,1),(0,2),(1,1),(1,2),(2,2),(3,3)\} \\
T^3 &= T^2
\end{aligned}$$

Consequently, the solutions set Z_w of $w(\mathscr{M}_1, \mathscr{M}_2, \mathscr{M}_3, x < 2, x = x + 1)$ is:

$$\begin{aligned}
Z_w &= \{(a,b) \mid (a,b) \in T^3 \wedge (x \mapsto b) \in sol(x \geq 2)\} \\
&= \{(0,2),(1,2),(2,2),(3,3)\}
\end{aligned}$$

Computing Z_w is undecidable in general as there is no termination guarantee of the iterating process. This is the reason why this computation is usually abstracted using abstract domain computation.

4.4 Abstracting the fixed point computation

Implementing the *join* operator mentionned above can be done by abstracting the computation of the fixed point within the Polyhedral abstract domain. Let $P^\sharp$ be a conjunction of linear restraints, the intersection of which defines a convex polyhedron, that over-approximates the set T. Hence, we can compute $P^\sharp$ as the least fixed point of:

$$P^{i+1} = \{(\mathscr{M}_k, \mathscr{M}_j) \mid (P^i \wedge \alpha_{poly}(\, w(\mathscr{M}_k, \mathscr{M}_j, \_, Dec, Body))) \sqcup P^i \quad (6)$$
$$P^0 = \{(\alpha_{poly}((\mathscr{M}_1, \mathscr{M}_1)) \quad (7)$$

Compared to eq. 4 and 5, the computation is realized in the abstract domain using α_{poly} the abstraction function of the Polyhedral abstract domain.
Let $Z_w^\sharp$ be the approximation of the set of solutions of w, obtained by application of α_{poly}:

$$Z_w^\sharp = \{(\mathscr{M}_i, \mathscr{M}_j) \mid (\mathscr{M}_i, \mathscr{M}_j) \in P^\sharp \wedge \mathscr{M}_j \in \alpha_{poly}(sol(\neg Dec))\}$$

Looking at the above example where $\mathscr{M}$ is just composed of the mapping of $x \mapsto v$, it is worth introducing different representations of the stores as we progress in the fixed point computation. When P^i is computed

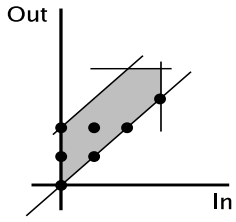

Figure 3: Exact and approximated fixed point

over x_k and establishes a relation in between stores $\mathscr{M}_k$ and $\mathscr{M}_j$ that contains x_j, we note: $P^i(x_k,x_j)$. If P^i is then considered over y_k,y_j, then we will simply write $P^i(y_k,y_j)$ and apply variable substitution. With these notations, we have the following computation:

$$\begin{array}{rcl}
P^0(x_{in},x_{out}) & = & x_{in} \geq 0 \wedge x_{in} \leq 3 \wedge x_{in} = x_{out} \\
P^1(x_{in},x_{out}) & = & (P^0(x_{in},x_0) \wedge x_0 \leq 1 \wedge x_{out} = x_0 + 1)_{x_{in},x_{out}} \sqcup P^0(x_{in},x_{out}) \\
 & = & (x_{in} \geq 0 \wedge x_{in} \leq 1 \wedge x_{out} = x_{in} + 1) \sqcup P^0(x_{in},x_{out}) \\
 & = & x_{in} \geq 0 \wedge x_{in} \leq 3 \wedge x_{out} \leq x_{in} + 1 \wedge x_{out} \geq x_{in} \\
P^2(x_{in},x_{out}) & = & (P^1(x_{in},x_1) \wedge x_1 \leq 1 \wedge x_{out} = x_1 + 1)_{x_{in},x_{out}} \sqcup P^1(x_{in},x_{out}) \\
 & = & (x_{in} \geq 0 \wedge x_{in} \leq 3 \wedge x_{in} \leq x_{out} - 1) \sqcup P^1(x_{in},x_{out}) \\
 & = & x_{in} \geq 0 \wedge x_{in} \leq 3 \wedge x_{out} \leq x_{in} + 2 \wedge x_{out} \geq x_{in} \wedge x_{out} \leq 4 \\
P^3(x_{in},x_{out}) & = & (P_2(x_{in},x_2) \wedge x_2 \leq 1 \wedge x_{out} = x_2 + 1)_{x_{in},x_{out}} \sqcup P^2(x_{in},x_{out}) \\
 & = & (x_{in} \geq 0 \wedge x_{in} \leq 3 \wedge x_{in} \leq x_{out} - 1) \sqcup P^2(x_{in},x_{out}) \\
 & = & P^2(x_{in},x_{out})
\end{array}$$

Fig. 3 illustrates the difference between the abstract fixed point and the approximate fixed point. Points in the figure correspond to the elements of T^3, while the grey zone represents the convex polyhedron defined by P^3.
An approximation of the solutions of $w(\mathscr{M}_1,\mathscr{M}_2,\mathscr{M}_3,x<2,x=x+1)$ is given by:

$$\begin{array}{rcl}
Q & = & P^3(x_1,x_3) \wedge x_3 \geq 2 \\
 & = & x_3 \geq 2 \wedge x_3 \leq 4 \wedge x_1 \leq x_3 \wedge x_1 \leq 3 \wedge x_1 \geq x_3 - 2
\end{array}$$

On the Polyhedral domain, convergence of the fixed point computation over $w(\mathscr{M}_1,\mathscr{M}_2,\mathscr{M}_3,x<2,x=x+1)$ can be enforced by using widening techniques. The computation of P^{k+1} is modified in order to use a widening operator ∇ [11]. Thus, we have:

$$P^{k+1} = P^k(Init,Out) \nabla (P^k \wedge \alpha_{poly}(w(\mathscr{M}_1,\mathscr{M}_2,\mathscr{M}_3,Dec,Body)))$$

A concrete algorithm for computing this approximation is given in [14], which permits to build implementation of w in a constraint solver. As rooted in the Abstract Interpretation domain, the relation w inherits from some of its fundamental correctness results, i.e., soundness and termination. However, it is worth pinpointing some differences.

Usually, a convex abstract polyhedron denotes the set of linear relations that hold over variables at a given point of a sequential program under analysis. As the goal here is to correctly approximate the set of solutions of a *w* relation, the polyhedron describes relations between input and output values and, thus, they involve more variables in the equations. In Abstract Interpretation, the analysis can be performed only once, whereas, in the case of the *w* relation, the *join* operation is launched everytime the relation is awaked without being succesfull in solving one of the guarded-constraint. As a consequence, we found out that it was not reasonable to use standard libraries to compute over polyhedra, such as PPL [2], because they use a *dual representation for Polyhedra*, which is a source of exponential time computations for the conversion.

4.5 Illustrative example

Looking at an iterative computation over unbounded domains as a relation captured by a *w* constraint operator is interesting for adressing Constraint-Based Reacheability problems. On the one hand, the suspension mechanism offered by constraint reasoning allows us to cope with the approximation problem, i.e., the set of states that is considered is determined by the informations existing in the constraint store, which makes the reasoning more accurate w.r.t. the property to be demonstrated. On the other hand, adding abstract domain computations to the *w* relation allows us to increase the level of deductions that can be achieved at each awakening of the *w* constraint operator. To illustrate this remark, consider the following *C* program:

```
f(  int i, ...  )  {
a.    j = 100;
b.      while( i > 0)
c.        { j=j+1 ; i=i-1 ;}
d.      ...
e. if( j > 500)
f.     ...
```

A typical reachability problem is to find out a value of `i` such that statement `f.` is executed. Existing approaches for solving this reachability problem consider a path passing through `f.`, e.g., `a-b-d-e-f`, and try to solve the *path condition* attached to this path. In this case, it means extracting constraint $j_1 = 100 \wedge i_1 \leq 0 \wedge j_1 > 500$ and solving it to show that the constraint system is unsatisfiable, i.e., the corresponding path is infeasible. Then, these approaches backtrack to select another path (e.g., `a-b-c-b-d-e-f` with path condition $j_1 = 100 \wedge i_1 > 0 \wedge j_2 = j_1 + 1 \wedge i_2 = i_1 - 1 \wedge i_2 \leq 0 \wedge j_2 > 500$) and repeat the process again, until a satisfiable path condition is found. This example is pathologic for these approaches, as only the paths that iterate more than 400 times in the loop will reach statement `f.`. Hopefully, using the constraint operator $w(\mathscr{M}_1, \mathscr{M}_2, \mathscr{M}_3, i > 0, j = j + 1 \wedge i = i - 1)$ permits us to unrool dynamically 400 times the loop without backtracking. The relational analysis performed on the Polyhedral abstract domain by the *w* operator determines that $j_{out} - i_{in} = 100$ whatever be the number of loop unrollings. Here, combining precise constraint reasoning in the concrete domain, with constraint extrapolation through abstract domain computations, offers us an efficient way of solving reachability problems on infinite-state systems.

5 Conclusions

In this paper, we have presented Constraint-Based Reachability as a process to combine constraint reasoning and abstraction techniques for solving reachability problems in infinite-state systems. The contribution is two-fold: first, we have revisited constraint consistency-filtering techniques by the prism of abstract domain computations ; second, we explained how to introduce abstract domain computation within the *w* constraint operator reasoning. We have illustrated these notions with several examples in order to ease the understanding of the reader.

This appraoch has been implemented and tested on several problems, including real-world programs [19, 20]. The goal is now to broader the scope of these techniques that combine constraint reasoning and abstraction techniques, to adress fundamental problems such as reachability in infinite-state systems.

Acknowledgements

We are indebted to Bernard Botella and Mireille Ducassé for fruitful discussions on earlier versions of this work.

Appendix

This appendix contains the proofs of some of the results stated in the paper.

Theorem 9 *Let p be a filtering operator associated with constraint c_i, then p computes domain-consistency iff $p = f^{\sharp}_{i\_arc}$.*

Proof 1 *($\Leftarrow$) Let $S_1 = (f_i \circ \gamma_{arc})(S)$. From the definitions of f_i and γ, we get that S_1 is the solution set of constraint c_i, given the initial domains S (we write $S_1 = sol(c_i, S)$). Hence, $S' = \alpha_{arc}(S_1) = (A_1, \ldots, A_m)$ with*
$A_k = \{x_k \mid x \in sol(c_i, S)\}$. So, c_i computes domain-consistency.
($\Rightarrow$) Let p be a domain-consistency filtering operator. Suppose that there exists S such that $p(S) = (A_1, \ldots, A_m)$ be strictly greater than $f^{\sharp}_{i\_arc}(S) = (B_1, \ldots, B_m)$. Then, there exists at least one k such as $A_k \supsetneq B_k$. Hence, there exists an element x_k of A_k that does not belong to any solution of constraint c_i. Hence, p cannot computes domain-consistency which is contradictory with the hypothesis. On the other side, p cannot be smaller than $f^{\sharp}_{i\_arc}$ as it means that the filtering operator removes solutions. Hence, if p computes domain-consistency then $p = f^{\sharp}_{i\_arc}$. □

Theorem 10 *If p is a filtering operator associated to constraint c_i, then p computes bound-consistency iff $p = f^{\sharp}_{i\_bound}$.*

Proof 2 *($\Leftarrow$) From theorem 9, given initial intervals I, the domains $f^{\sharp}_{i\_arc} \circ \gamma_{inter}(I)$ are domain-consistent for constraint c_i. Applying function α_{inter} is similar to the process that keeps extremal values of each element of $f^{\sharp}_{i\_arc} \circ \gamma_{inter}(I)$. Hence, the resulting intervals satisfy the bound-consistency property.*
($\Rightarrow$) (similar to the proof of theorem 9) If the filtering operator p is greater than $f^{\sharp}_{i\_bound}$, then the computed intervals contain at least one bound that is not part of a solution of c_i, violating so the bound-consistency property. On the contrary, by supposing that p is smaller than $f^{\sharp}_{i\_bound}$ then solutions are lost and p is no more a filtering operator. Hence, if p is a filtering operator guaranteeing bound-consistency then $p = f^{\sharp}_{i\_bound}$. □

References

[1] K. Apt (1999): *The Essence of Constraint Propagation*. *Theoretical Computer Science* 221(1-2), pp. 179–210, doi:10.1016/S0304-3975(99)00032-8.

[2] Roberto Bagnara, Elisa Ricci, Enea Zaffanella & Patricia M. Hill (2002): *Possibly Not Closed Convex Polyhedra and the Parma Polyhedra Library*. In Springer, editor: *SAS'02: In M. V. Hermenegildo and G. Puebla, editors, Proc. of the Static Analysis Symposium, LNCS* 2477, pp. 213–229, doi:10.1007/3-540-45789-5_17.

[3] S. Bardin & P. Herrmann (2011): *OSMOSE: Automatic Structural Testing of Executables*. *Software Testing, Verification and Reliability (STVR)* 21(1), pp. 29–54, doi:10.1002/stvr.423.

[4] Peter Boonstoppel, Cristian Cadar & Dawson Engler (2008): *RWset: Attacking path explosion in constraint-based test generation*. In: *Int. Conference on Tools and Algorithms for the Constructions and Analysis of Systems (TACAS'08)*, pp. 351–366, doi:10.1007/978-3-540-78800-3_27.

[5] B. Botella, A. Gotlieb & C. Michel (2006): *Symbolic execution of floating-point computations*. *The Software Testing, Verification and Reliability journal* 16(2), pp. pp 97–121, doi:10.1002/stvr.333.

[6] Richard H. Carver (1996): *Testing abstract distributed programs and their implementations: A constraint-based approach*. *Journal of Systems and Software* 33(3), pp. 223–237, doi:10.1016/0164-1212(96)00024-6.

[7] F. Charreteur, B. Botella & A. Gotlieb (2009): *Modelling dynamic memory management in Constraint-Based Testing*. *The Journal of Systems and Software* 82(11), pp. 1755–1766. Special Issue: TAIC-PART 2007 and MUTATION 2007, doi:10.1016/j.jss.2009.06.029.

[8] F. Charreteur & A. Gotlieb (2010): *Constraint-Based Test Input Generation for Java Bytecode*. In: *Proc. of the 21st IEEE Int. Symp. on Softw. Reliability Engineering (ISSRE'10)*, San Jose, CA, USA, doi:10.1109/ISSRE.2010.26.

[9] Chiu Wo Choi, Warwick Harvey, J. H. M. Lee & Peter J. Stuckey (2006): *Finite Domain Bounds Consistency Revisited*. In: *Australian Conference on Artificial Intelligence*, pp. 49–58, doi:10.1007/11941439_9.

[10] P. Cousot & R. Cousot (1977): *Abstract Interpretation : A unified lattice model for static analysis of programs by construction or approximation of fixpoints*. In: *Proceedings of Symp. on Principles of Programming Languages*, ACM, pp. 238–252, doi:10.1145/512950.512973.

[11] P. Cousot & N. Halbwachs (1978): *Automatic Discovery of Linear Restraints Among Variables of a Program*. In: *Proceedings of Symp. on Principles of Programming Languages*, ACM, pp. 84–96, doi:10.1145/512760.512770.

[12] Patrick Cousot & Radhia Cousot (1976): *Static Determination of dynamic properties of programs*. In: *Proc. of the 2nd International Symp. on Programming*, Dunod, pp. 106–130.

[13] Giorgio Delzanno & Andreas Podelski (2001): *Constraint-based deductive model checking*. *International Journal on Software Tools for Technology Transfer (STTT)* 3(3), pp. 250–270, doi:10.1007/s100090100049.

[14] T. Denmat, A. Gotlieb & M. Ducasse (2007): *An Abstract Interpretation Based Combinator for Modeling While Loops in Constraint Programming*. In: *Proceedings of Principles and Practices of Constraint Programming (CP'07)*, Springer Verlag, LNCS 4741, Providence, USA, pp. 241–255, doi:10.1007/978-3-540-74970-7_19.

[15] T. Denmat, A. Gotlieb & M. Ducasse (2007): *Improving Constraint-Based Testing with Dynamic Linear Relaxations*. In: *18th IEEE International Symposium on Software Reliability Engineering (ISSRE' 2007)*, Trollhttan, Sweden, doi:10.1109/ISSRE.2007.12.

[16] Cormac Flanagan (2004): *Automatic software model checking via constraint logic*. *Sci. Comput. Program.* 50(1-3), pp. 253–270. Available at `http://dx.doi.org/10.1016/j.scico.2004.01.006`.

[17] Gordon Fraser & Franz Wotawa (2007): *Test-Case Generation and Coverage Analysis for Nondeterministic Systems Using Model-Checkers*. In: *ICSEA*, p. 45, doi:10.1109/ICSEA.2007.71.

[18] P. Godefroid, N. Klarlund & K. Sen (2005): *DART: directed automated random testing*. In: *Proc. of PLDI'05*, pp. 213–223, doi:10.1145/1064978.1065036.

[19] A. Gotlieb (2009): *EUCLIDE: A Constraint-Based Testing platform for critical C programs*. In: *2th IEEE International Conference on Software Testing, Validation and Verification (ICST'09)*, Denver, CO, doi:10.1109/ICST.2009.10.

[20] A. Gotlieb (2012): *TCAS software verification using Constraint Programming*. *The Knowledge Engineering Review* 27(3), pp. 343–360, doi:10.1017/S0269888912000252.

[21] A. Gotlieb, B. Botella & M. Rueher (1998): *Automatic Test Data Generation Using Constraint Solving Techniques*. In: *Proc. of Int. Symp. on Soft. Testing and Analysis (ISSTA'98)*, pp. 53–62, doi:10.1145/271771.271790.

[22] A. Gotlieb, B. Botella & M. Rueher (2000): *A CLP Framework for Computing Structural Test Data*. In: *Proceedings of Computational Logic (CL'2000)*, LNAI 1891, London, UK, pp. 399–413, doi:10.1007/3-540-44957-4_27.

[23] A. Gotlieb, T. Denmat & B. Botella (2005): *Constraint-based test data generation in the presence of stack-directed pointers*. In: *20th IEEE/ACM International Conference on Automated Software Engineering (ASE'05)*, Long Beach, CA, USA. 4 pages, doi:10.1145/1101908.1101958.

[24] A. Gotlieb, T. Denmat & B. Botella (2007): *Goal-oriented test data generation for pointer programs*. *Information and Soft. Technol.* 49(9-10), pp. 1030–1044, doi:10.1016/j.infsof.2006.10.016.

[25] T. Henzinger, R. Jhala, R. Majumdar & G. Sutre (2003): *Software verification with Blast*. In: *Proc. of 10th Workshop on Model Checking of Software (SPIN)*, pp. 235–239, doi:10.1007/3-540-44829-2_17.

[26] K. Marriott & P.J. Stuckey (1998): *Programming with Constraints : An Introduction*. The MIT Press.

[27] R.A. Moore (1966): *Interval Analysis*. Prentice Hall, New Jersey.

[28] M.N. Wegman & F.K. Zadeck (1991): *Constant Propagation with Conditional Branches*. *ACM Transactions on Programming Language and Systems* 13(2), pp. 181–210, doi:10.1145/103135.103136.

[29] N. Williams, B. Marre, P. Mouy & M. Roger (2005): *PathCrawler: Automatic Generation of Path Tests by Combining Static and Dynamic Analysis*. In: *Proc. Dependable Computing - EDCC'05*, doi:10.1007/11408901_21.